Machine Learning
An Artificial Intelligence Approach
Michalski, Carbonell, Mitchell.
1983

15

LEARNING EFFICIENT CLASSIFICATION PROCEDURES AND THEIR APPLICATION TO CHESS END GAMES

J. Ross Quinlan
The Rand Corporation

ABSTRACT

A series of experiments dealing with the discovery of efficient classification procedures from large numbers of examples is described, with a case study from the chess end game king-rook versus king-knight. After an outline of the inductive inference machinery used, the paper reports on trials leading to correct and very fast attribute-based rules for the relations lost 2-ply and lost 3-ply. On another tack, a model of the performance of an idealized induction system is developed and its somewhat surprising predictions compared with observed results. The paper ends with a description of preliminary work on the automatic specification of relevant attributes.

15.1 INTRODUCTION

This paper reports on experiments that recover valuable information from large masses of low-grade data by a process of inductive inference. The data are relatively unstructured examples and counterexamples of a concept of considerable complexity. The information that is sought is a means of identifying examples of the concept or, in other words, a classification rule. The distinguishing characteristics of this work are the large numbers of examples employed in forming the concepts and the computational efficiency of the classification rules discovered in this way as compared with other classification methods for the same task. (In one case, the classification rule is five times as fast as the best alternative method that I could devise.)

The concepts to be developed have been drawn from the chess end game king-rook versus king-knight, which can be difficult even for masters [Kopec & Niblett, 1980]. This end game has proved to be an excellent testing ground, providing classification tasks of a range of difficulties and a large underlying universe of more than a million possible configurations of pieces. However, it should be noted that the inductive inference machinery that has been developed is in no way tied to this application, and is currently being used by other workers for a different aspect of chess [Shapiro & Niblett, 1982] and in a medical domain [Bratko & Mulec, 1981].

The induction algorithm used for this project is called ID3. ID3 takes objects of a known class described in terms of a fixed collection of properties or attributes, and produces a decision tree over these attributes that correctly classifies all the given objects. Two qualities differentiate it from other general-purpose inference systems such as INDUCE [Michalski, 1980], SPROUTER [Hayes-Roth & McDermott, 1977] and THOTH-P [Vere, 1978]. The first concerns the way that the effort required to accomplish an induction task grows with the difficulty of that task. ID3 was specifically designed to handle masses of objects, and in fact its computation time increases only linearly with difficulty as modeled by the product of:

- the number of given exemplary objects,
- the number of attributes used to describe objects, and
- the complexity of the concept to be developed (measured by the number of nodes on the decision tree)

[Quinlan, 1979a]. On the negative side, this linearity is purchased at the cost of descriptive power. The concepts developed by ID3 can only take the form of decision trees based on the attributes given, and this "language" is much more restrictive than the first-order or multivalued logic in which the above systems express concepts. [Dietterich & Michalski, 1979] gives an analysis and survey of inductive inference methodologies.

The main body of this report contains four sections. The first, Section 15.2, introduces ID3 as a descendant of Hunt's Concept Learning System (CLS). Section 15.3 summarizes applications of ID3 to discovering decision trees for the relations "knight's side is lost n-ply" where n is 2 or 3; detailed accounts appear in [Quinlan, 1979b; Quinlan, 1980]. The last two sections deal with recent work along different dimensions. Section 15.4 considers the question of discovering approximate rather than exact rules. It develops a model of how an idealized induction system might behave when shown only a fraction of all possible objects, and compares the predictions of this model to observed results. Section 15.5 tackles the problem of defining features used to describe objects, and outlines techniques aimed at automating the discovery of the attributes themselves.

15.2 THE INDUCTIVE INFERENCE MACHINERY

ID3, a descendant of Hunt's CLS [Hunt *et al.*, 1966], is a comparatively simple mechanism for discovering a classification rule for a collection of objects belonging to two classes. As mentioned above, each object must be described in terms of a fixed set of attributes, each of which has its own (small) set of possible *attribute values*. As an illustration, "color" and "baud rate" might be attributes with sets of possible values {red,green,blue} and {110,300,1200,2400,4800} respectively.

A classification rule in the form of a decision tree can be constructed for any such collection C of objects. If C is empty then we associate it arbitrarily with either class. If all objects in C belong to the same class, then the decision tree is a leaf bearing that class name. Otherwise C contains representatives of both classes; we select an attribute and partition C into disjoint sets C_1, C_2, ... , C_n where C_i contains those members of C that have the i^{th} value of the selected attribute. Each of these subcollections is then handled in turn by the same rule-forming strategy. The eventual outcome is a tree in which each leaf carries a class name, and each interior node specifies an attribute to be tested with a branch corresponding to each possible value of that attribute.

To illustrate this process, consider the collection C below. Each object is described in terms of three attributes: "height" with values {tall,short}, "hair" with values {dark,red,blond} and "eyes" with values {blue,brown}, and is followed by '+' or '−' to indicate the class to which it belongs.

C =	short,blond,blue: +	short,dark,blue: −	tall,dark,brown: −
	tall,blond,brown: −	tall,dark,blue: −	short,blond,brown: −
	tall,red,blue: +	tall,blond,blue: +	

If the second attribute is selected to form the root of the decision tree, this yields the tree shown in Figure 15-1. The subcollections corresponding to the values 'dark' and 'red' contain objects of only a single class, and so require no further work. If we select the third attribute to test for the 'blond' branch, this yields the tree in Figure 15-2. Now all subcollections contain objects of one class, so we can replace each subcollection by the class name to get the decision tree shown in Figure 15-3.

An object is classified by starting at the root of the decision tree, finding the value of the tested attribute in the given object, taking the branch appropriate to that value, and continuing in the same fashion until a leaf is reached. Notice that classifying a particular object may involve evaluating only a small number of the attributes depending on the length of the path from the root of the tree to the appropriate leaf. In the above case, the first step is always to inquire about the value of an object's "hair" attribute. If this value is "dark" or "red" the object can be classified immediately without looking at its other attributes. If the value is 'blond' then we must determine its value of "eyes" before classifying it. We never need to determine the value of the "height" attribute.

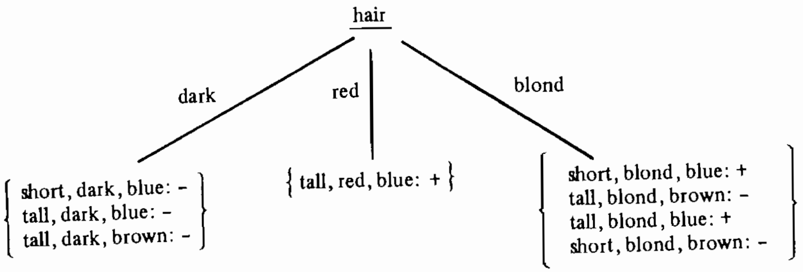

Figure 15-1: One-level decision tree.

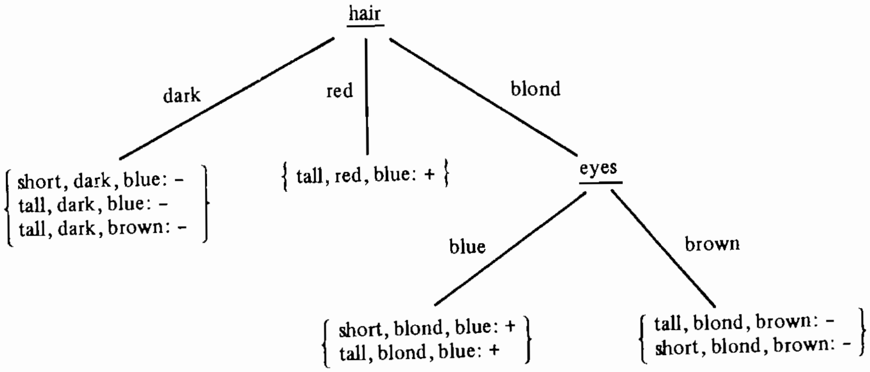

Figure 15-2: Two-level decision tree.

This rule-forming procedure will always work provided that there are not two objects belonging to different classes but having identical values for each attribute; in such cases the attributes are inadequate for the classification task. However, it is generally desirable that the tree be able to classify objects which were not used in its construction, and so the leaves corresponding to an empty set of examples (where a class is chosen randomly) should be kept to a minimum. If we adopted the simple-minded algorithm:

> "Select the first attribute for the root of the tree, the
> second attribute for the next level, and so on."

the result would tend towards the complete tree with a leaf for each point in the attribute space—clearly an unsatisfactory situation. The whole skill in this style of induction lies in selecting a useful attribute to test for a given collection of objects so that the final tree is in some sense minimal. Hunt's work used a lookahead scheme driven by a system of measurement and misclassification costs in an attempt to get minimal-cost trees. ID3 uses an information-theoretic approach aimed at minimizing the expected number of tests to classify an object

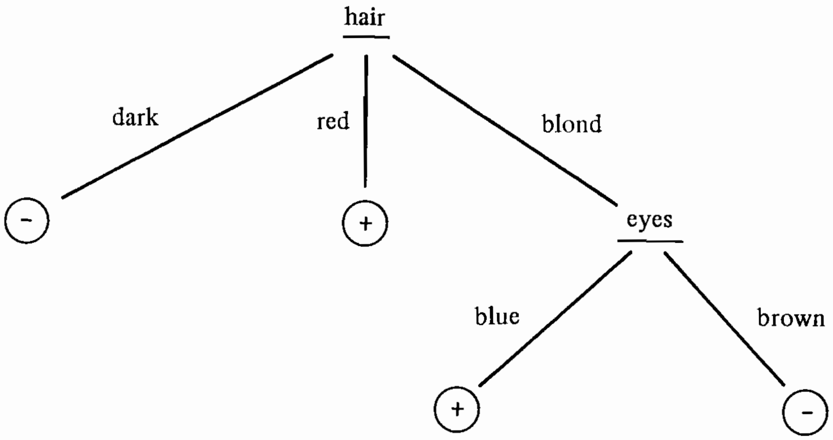

Figure 15-3: Decision tree with class names.

A decision tree may be regarded as an information source that, given an object, generates a message which is the class of that object ("plus" or "minus", say). The attribute selection part of ID3 is based on the plausible assumption that the complexity of the decision tree is strongly related to the amount of information conveyed by this message. If the probability of these messages is p^+ and p^- respectively, the expected information content of the message is

$$-p^+ \log_2 p^+ - p^- \log_2 p^-$$

With a known set C of objects we can approximate these probabilities by relative frequencies, so that p^+ becomes the proportion of objects in C with class "plus". So we will write M(C) to denote this calculation of the expected information content of a message from a decision tree for a set C of objects, and define M({}) = 0. Now consider as before the possible choice of A as the attribute to test next. The partial decision tree is shown in Figure 15-4. The values A_i of attribute A are mutually exclusive (even though different attributes may not be), so the new expected information content will be:

$$B(C,A) = (\text{probability that value of } A \text{ is } A_i) \times M(C_i)$$

where again we can replace the probabilities by relative frequencies. The suggested choice of attribute to test next is that which gains the most information, in other words, for which

$$M(C) - B(C,A)$$

is maximal.

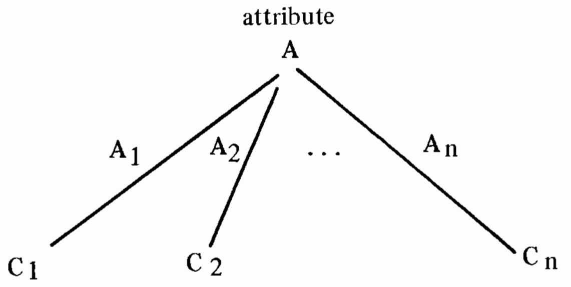

Figure 15-4: Partial decision tree.

To illustrate the idea, consider the choice of the first attribute to test from the example given earlier. The collection C of objects contains 3 in class '+' and 5 in '−', so

$$M(C) = -\ 3/8 \log_2 3/8 - 5/8 \log_2 5/8$$
$$= 0.954 \text{ bits}$$

Testing the first attribute gives the results shown in Figure 15-5. The information still needed for a rule for the "tall" branch is:

$$-\ 2/5 \log_2 2/5 - 3/5 \log_2 3/5 = 0.971 \text{ bits}$$

and for the "short" branch:

$$-\ 1/3 \log_2 1/3 - 2/3 \log_2 2/3 = 0.918 \text{ bits}$$

Thus the expected information content:

$$B(C, \text{“height”}) = 5/8 * 0.971 + 3/8 * 0.918$$
$$= 0.951 \text{ bits}$$

The information gained by testing this attribute is:

$$0.954 - 0.951 = 0.003 \text{ bits}$$

which is negligible. The tree arising from testing the second attribute was given previously. The branches for "dark" (with 3 objects) and "red" (1 object) require no further information, while the branch for "blond" contained 2 "plus" and 2 "minus" objects and so requires 1 bit. We have:

$$B(C, \text{“hair”}) = 3/8 * 0 + 1/8 * 0 + 4/8 * 1$$
$$= 0.5 \text{ bits}$$

and the information gained by testing "hair" is 0.954 − 0.5 = 0.454 bits. In a similar way the information gained by testing "eyes" comes to 0.347 bits. Thus the principle of maximizing expected information gain would lead ID3 to select "hair" as the attribute to form the root of the decision tree.

The procedure described above for constructing decision trees assumes that counting operations on the set of objects C (such as determining the number of "plus" objects with value A_i of attribute A) can be performed efficiently, which means in practice that C has to be kept in fast memory. What happens if the size of C precludes this? One way around the difficulty is given by the version space strategy [Mitchell, 1979] in which C is digested one object at a time. Such an approach depends on maintaining two sets S and G of maximally specific and maximally general rules that could account for all objects seen so far; these sets delimit all possible correct rules. However, when the rule is a decision tree over a large attribute space, the sizes of S and G will also become unmanageable. The line taken in ID3 is an iterative one which forms a succession of decision trees of (hopefully) increasing accuracy, until one is found that is entirely accurate. The method can be summarized as:

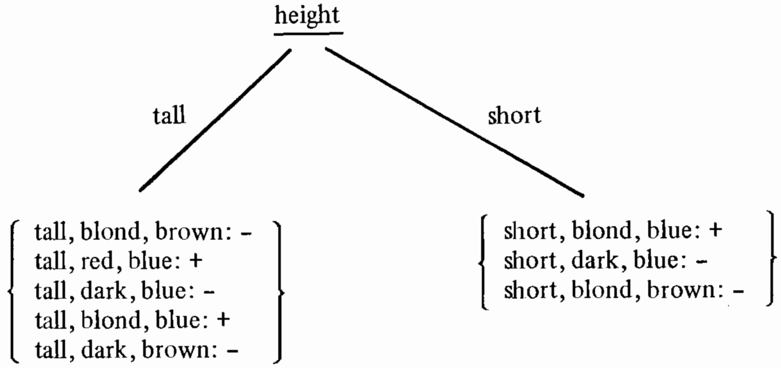

Figure 15-5: Binary attribute discrimination.

- select at random a subset of the given instances (called the *window*)
- *repeat*
    - ○ form a rule to explain the current window
    - ○ find the exceptions to this rule in the remaining instances
    - ○ form a new window from the current window and the exceptions to the rule generated from it

  until there are no exceptions to the rule

The process ends when a rule is formed that has no exceptions and so is correct for all of C. Two different ways of forming a new window have been tried. In the first, the current window is enlarged by the addition of up to some specified number of exceptions, and so the window grows. The second method attempts to identify "key" objects in the current window and replace the rest with exceptions, thus keeping the window size constant. Both methods were explored in trials with a non-trivial classification problem involving 14 attributes and nearly 2,000 objects for which a correct decision tree contained 48 nodes [Quinlan, 1979a]. The main findings were:

- The methods converge rapidly; typically only 4 iterations were required to find a correct decision tree.
- It was possible to develop a correct tree from a final window containing only a small fraction of the 2,000 objects.
- The process was not very sensitive to parameters such as the initial window size.
- The time to obtain a correct decision tree for a classification problem increases linearly with the difficulty of the problem as defined by the simple model given in the introduction.

These features, particularly the last, have enabled ID3 to discover correct decision trees for some very large classification problems.

15.3 THE LOST N-PLY EXPERIMENTS

One application of ID3 has been to discover classification rules for part of the end game (white) king-rook versus (black) king-knight. The relations completed are "knight's side is lost (in at most) n-ply" for n = 2 and n = 3; the 4-ply case is currently being tackled. The formal definition of "lost n-ply" is:

1. A black-to-move position is lost 0-ply if and only if
   a. the king is in checkmate, or
   b. the knight has been captured, the position is not stalemate, the white rook has not been captured and the black king cannot retaliate by capturing it.
2. A white-to-move position is lost n-ply (n odd) iff there is a white move giving a position that is lost n−1 ply.
3. A black-to-move position is lost n-ply (n even) iff all possible black moves give positions that are lost n−1 ply.

These definitions ignore the repetition and 50-move rules of chess, but are quite accurate for small values of n.

The obvious question is, why bother looking for classification rules when simple algorithms such as minimax will decide whether a position is lost n-ply? The answer is that a decision tree will classify a position in terms of its properties rather than by exploring the game tree. If attributes can be found that are adequate for this classification task and that are also relatively cheap to compute, then the classification of a position in terms of these attributes might well be faster than the minimax search of the game tree.

There are more than 11 million ways of placing the four pieces to form a legal black-to-move position; the corresponding figure for white-to-move is more than 9 million. (The difference arises because, for instance, the white king cannot be in check in a black-to-move position.) These counts include many symmetric variants of essentially the same position, and when these are removed the

numbers become approximately 1.8 million and 1.4 million, respectively. About 69,000 of the 1.8 million black-to-move positions are lost 2-ply, while roughly 474,000 of the 1.4 million white-to-move positions are lost 3-ply.

The first attempt at the lost 2-ply relation was made with a set of 25 attributes. 18 of these were low-level geometric properties of a position, such as:

> the distance from the black king to the knight

with values "1", "2" and ">2" king-moves, and:

> the two kings are on opposite sides of and next to a row or column occupied by the rook

with values "true" and "false". The remaining 7 attributes were somewhat higher-level and involved more computation, for example:

> the only move the black king can make creates a mating square for the rook

The attribute space with $3^6 \times 2^{19}$ points was much larger than the number of black-to-move positions. However, many different positions led to the same vector of attribute values; and, in fact, the 1.8 million positions dwindled to just under 30,000 distinct vectors. An implementation of ID3 coded in Pascal for a DEC KL-10 found a correct decision tree of 334 nodes in 144 seconds.

A second attempt was made on this problem in order to try out a different style of attribute, and to remove a minor inadequacy of the first set of attributes affecting a handful of positions. Instead of being for the most part low-level and geometric, the new attributes were all high-level, truth-valued features signaling key patterns of pieces on the board. Each of the 23 attributes was meant to capture some important mechanism of the lost 2-ply classification task. For example, one of the attributes took the value "true" if the position was of one of the forms shown in Figure 15-6. This attribute was intended to detect some situations in which the black king cannot move safely. As expected, these new attributes were more directly pertinent to the classification problem than were their geometric predecessors. They had the effect of compressing all possible positions into a smaller set of 428 distinct vectors. The same implementation of ID3 found a decision tree containing 83 nodes in less than 3 seconds.

These trials resulted in two decision trees for classifying an arbitrary black-to-move position as lost 2-ply or not. Their performance was then compared to two other means of arriving at the same classification. The first of these was the minimax search mentioned previously, which simply mirrors the definition of lost n-ply. The second was a "smarter" classification method called *specialized search* which examines only part of the game tree. For instance, to determine whether a position is lost 1-ply we only have to consider white moves that capture the knight or white rook moves to the edge of the board (for a possible mate). Specialized search thus employs domain knowledge; it is harder to write and debug than minimax, but is much faster.

For the purposes of comparison, all methods were implemented in Pascal.